

Fig. 1.

2/33

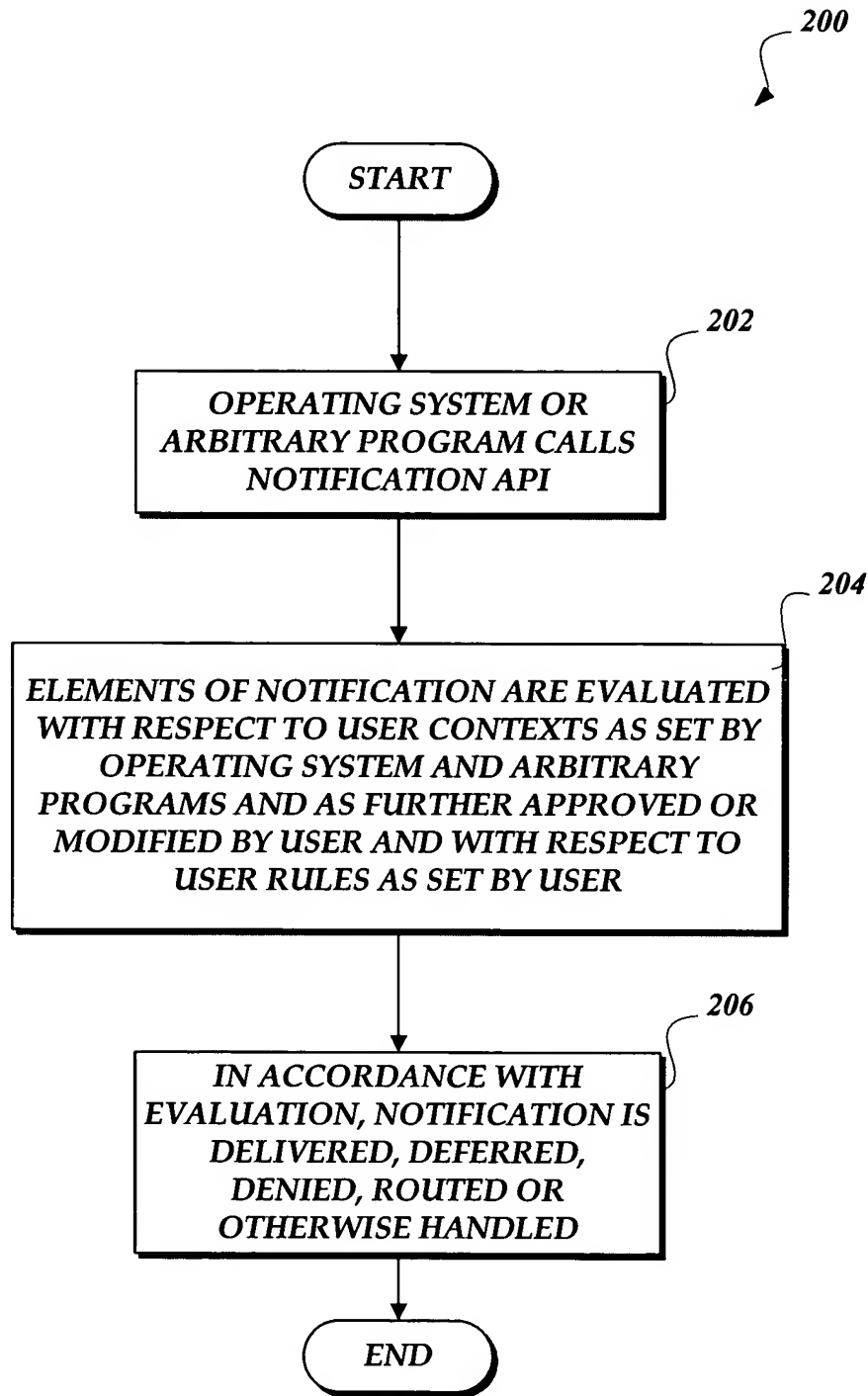


Fig.2.

3/33

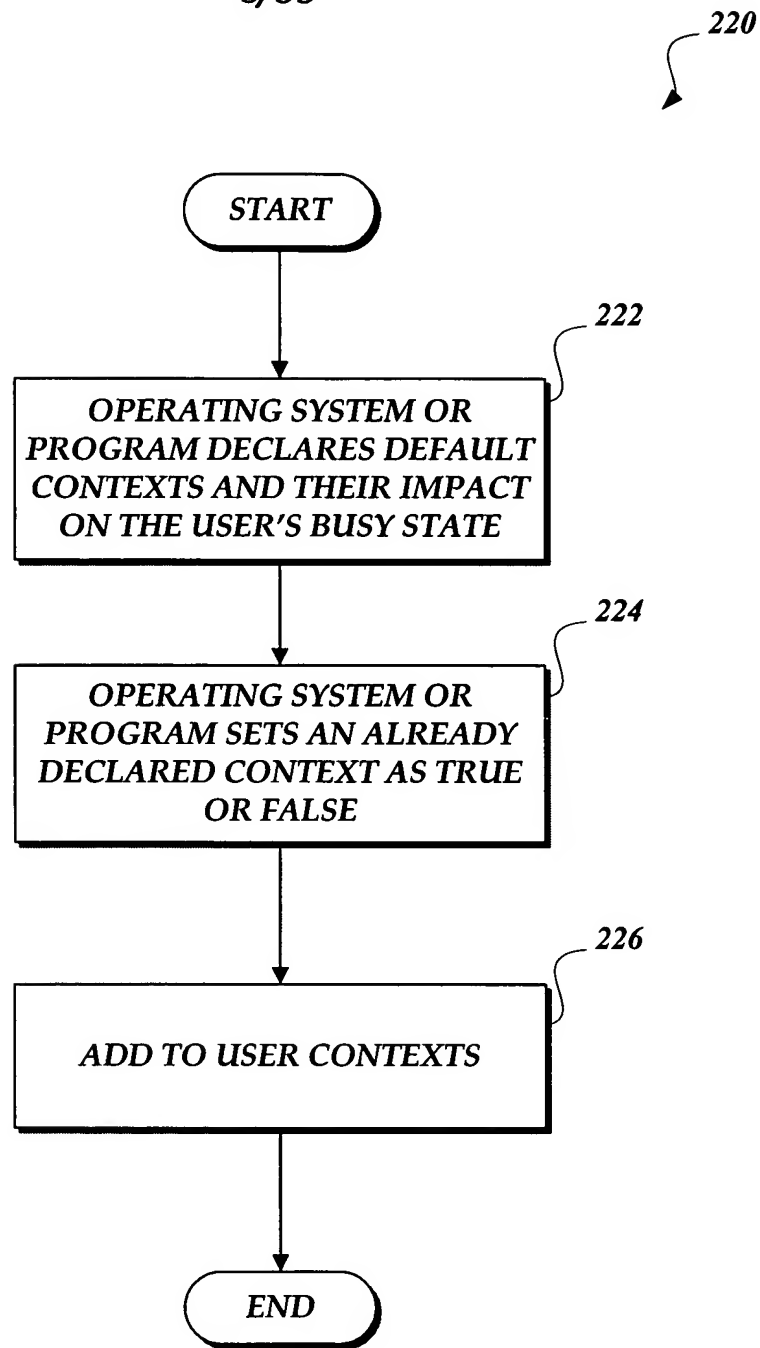


Fig.3.

4/33

230

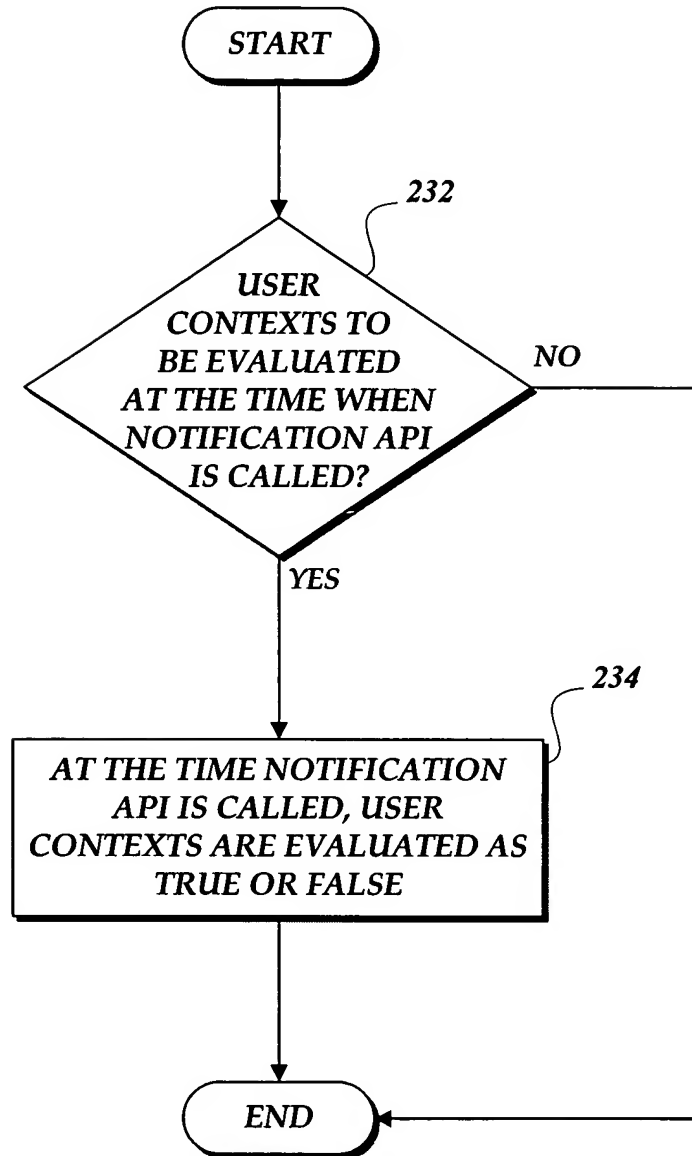


Fig.4.

5/33

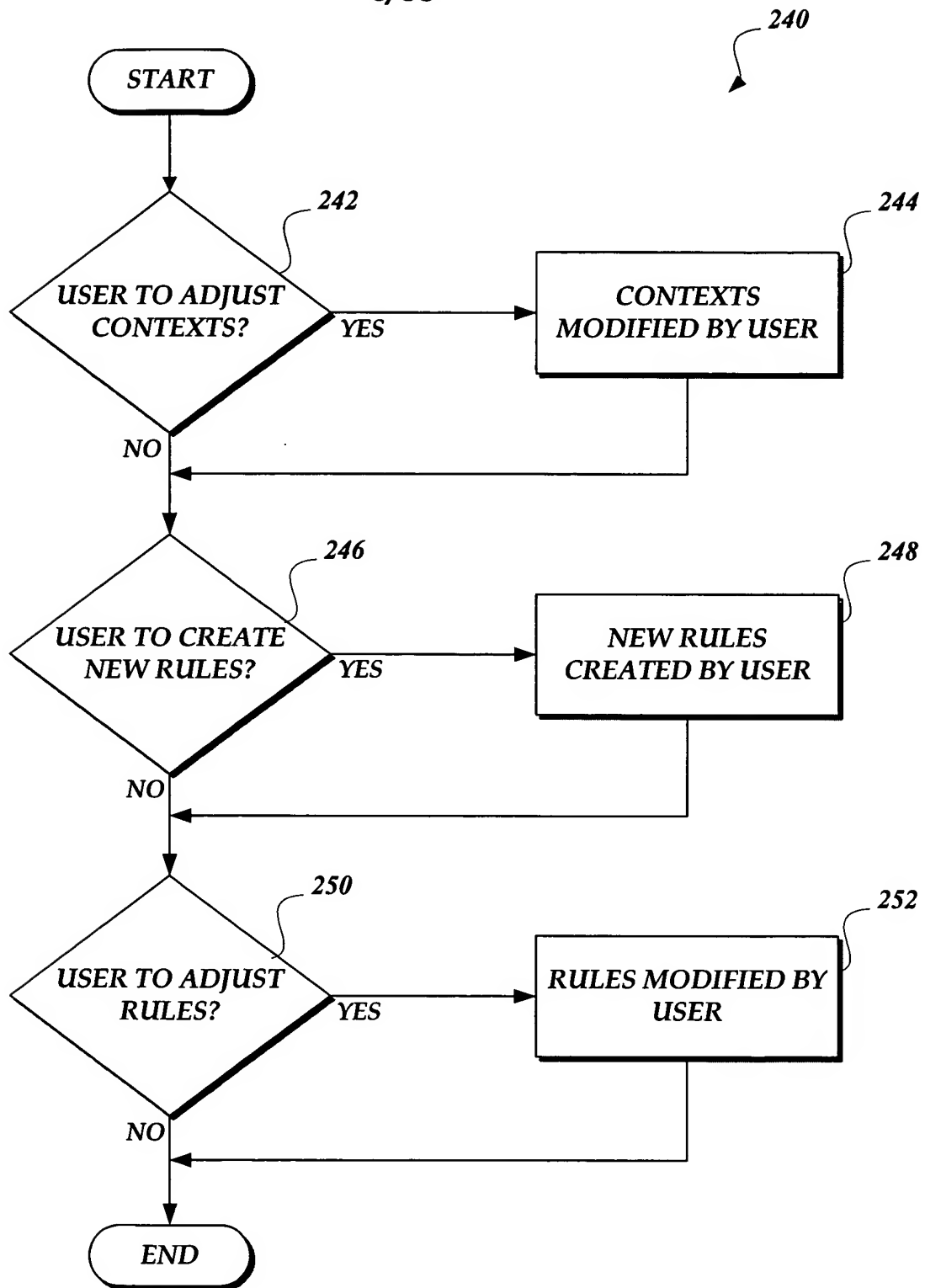


Fig.5.

6/33

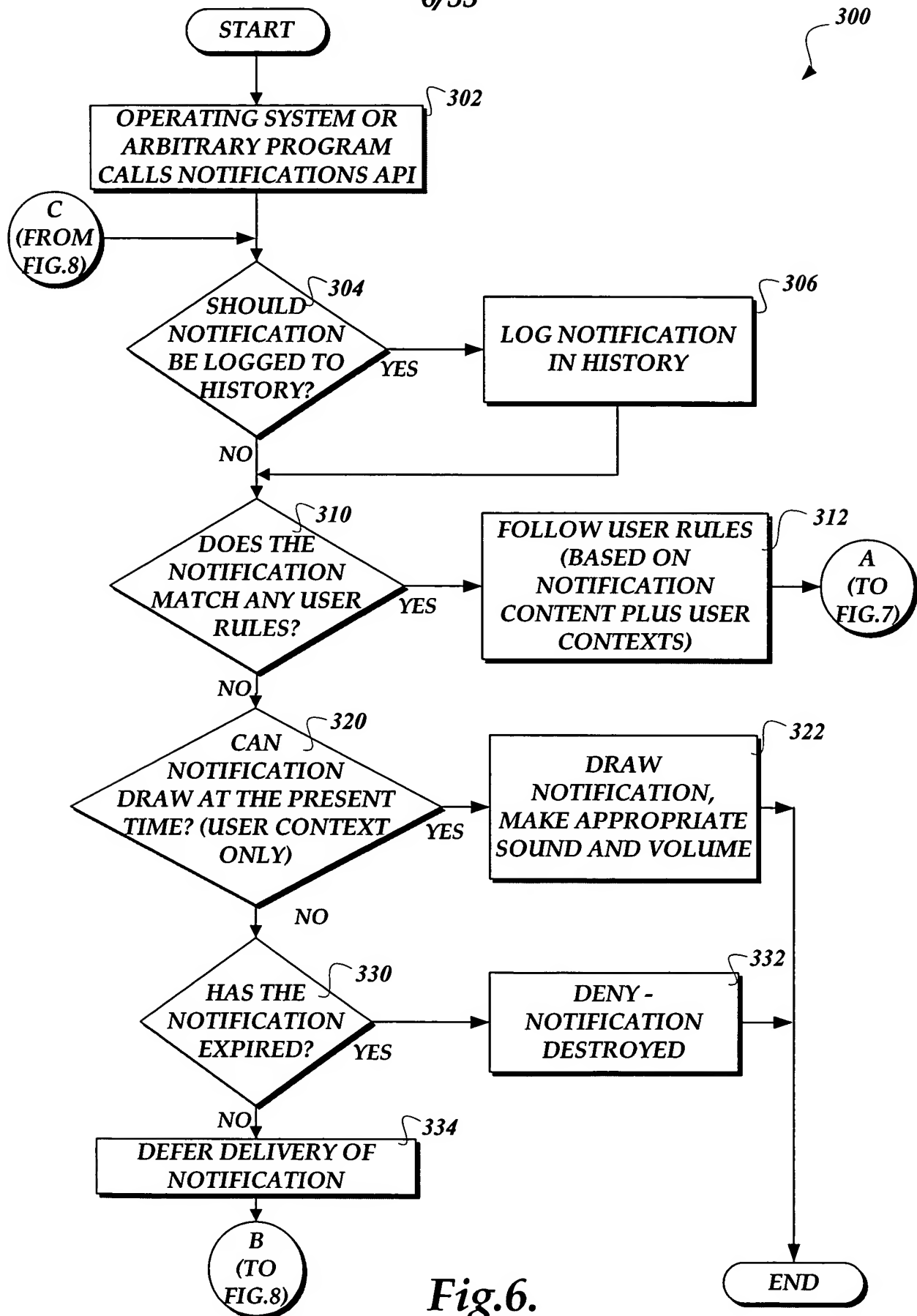


Fig.6.

7/33

350

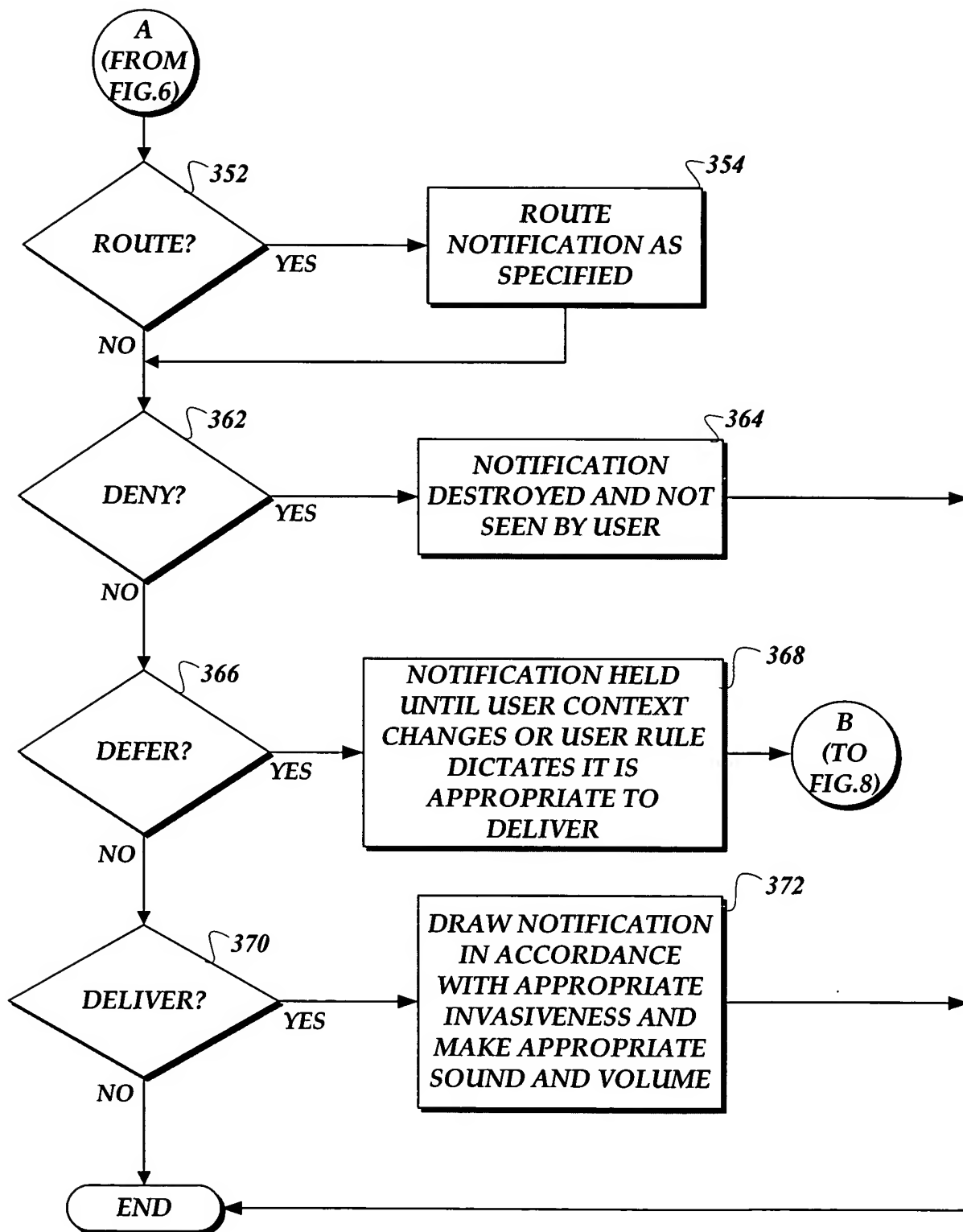


Fig.7.

8/33

380

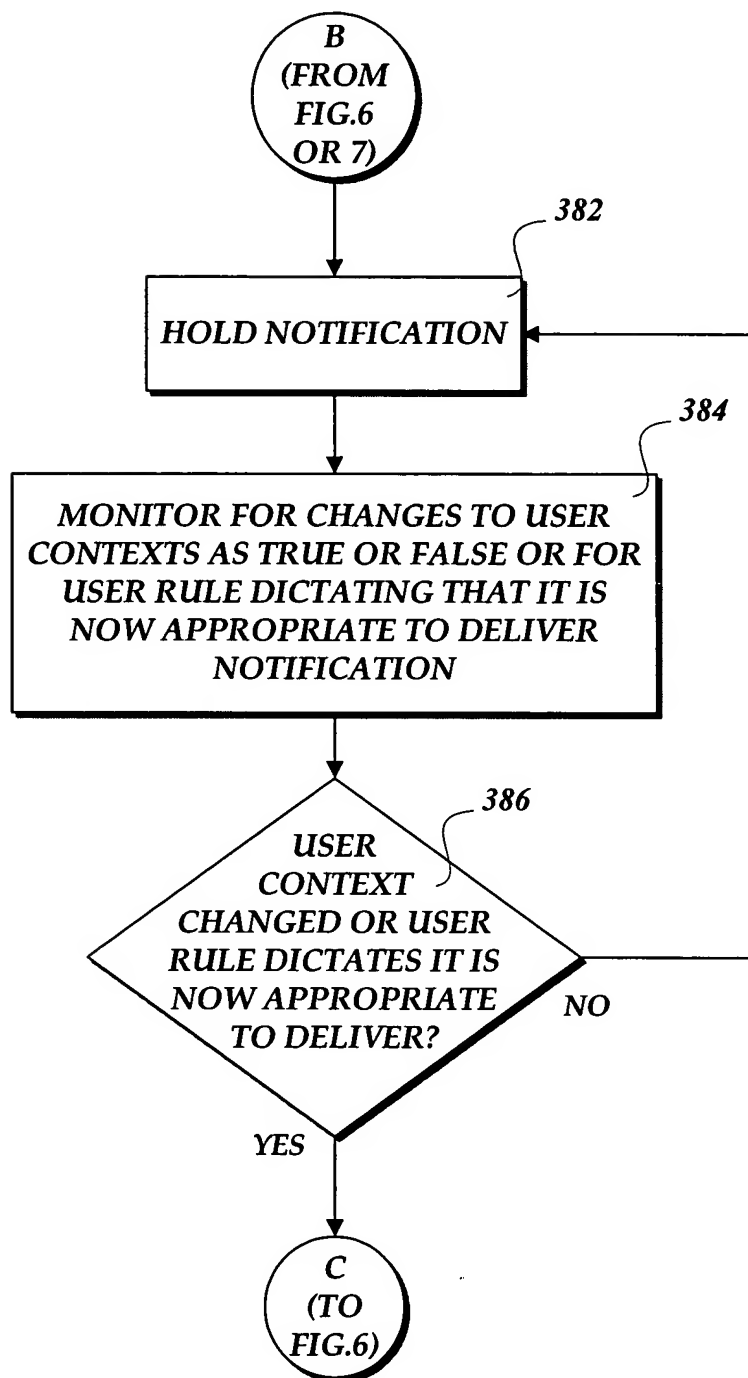


Fig.8.

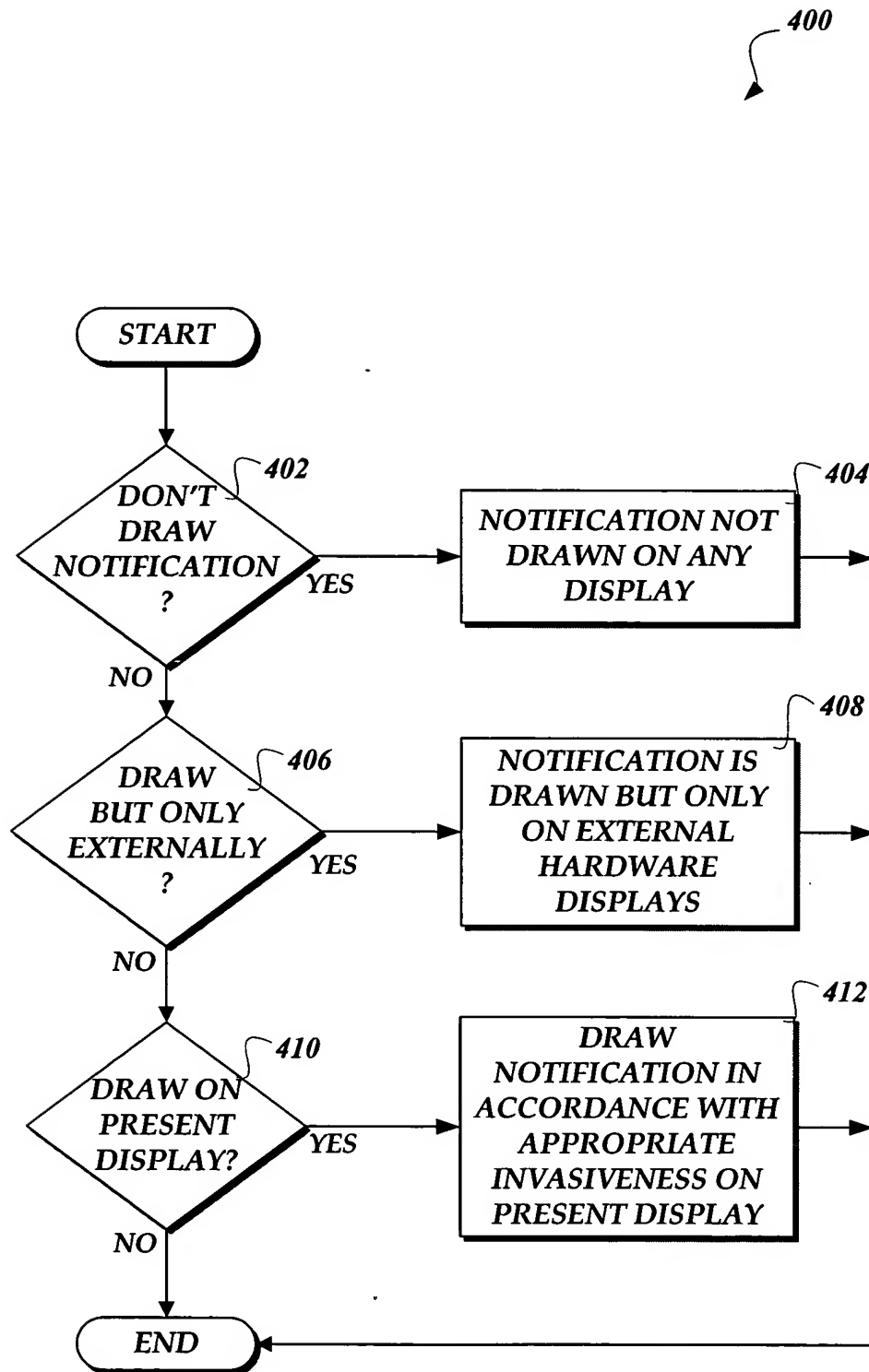


Fig.9.

10/33

420

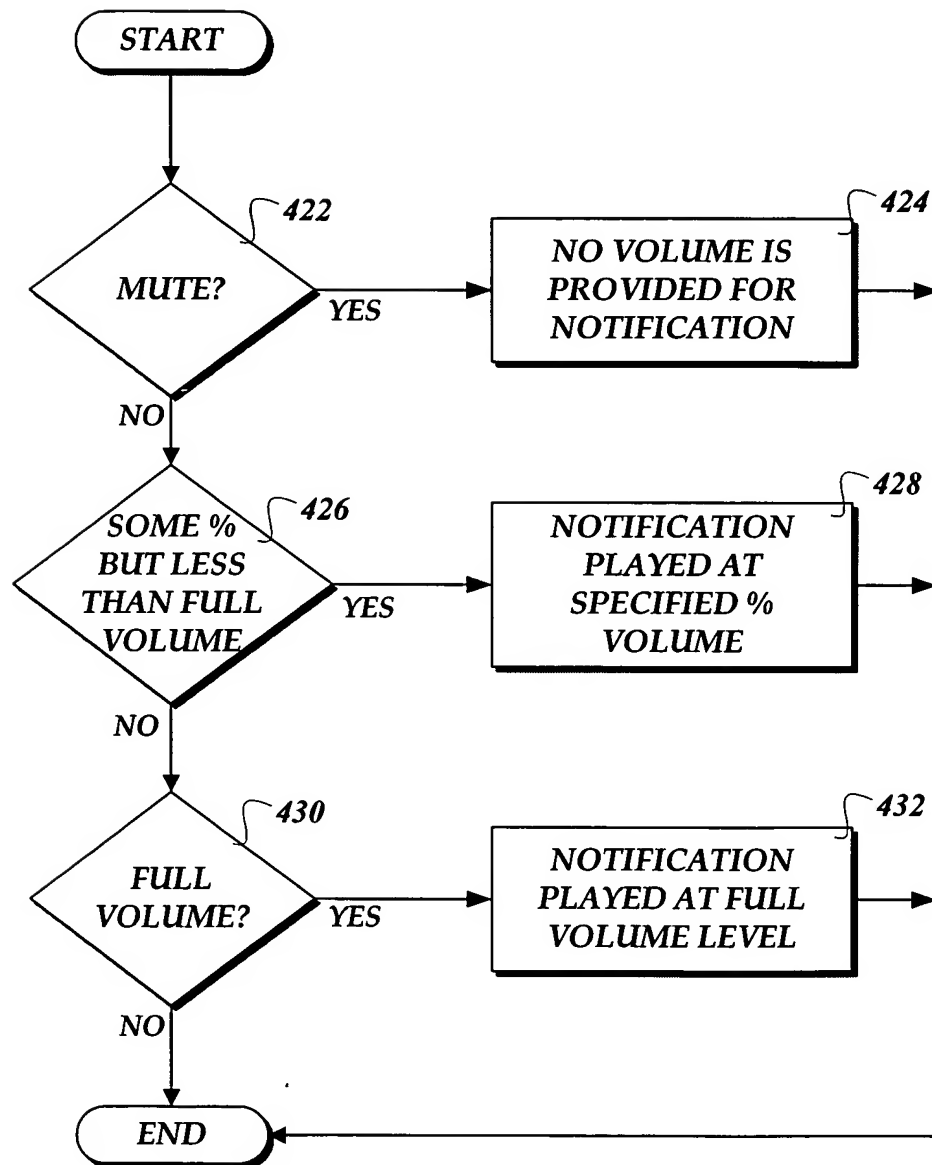


Fig.10.

11/33

500

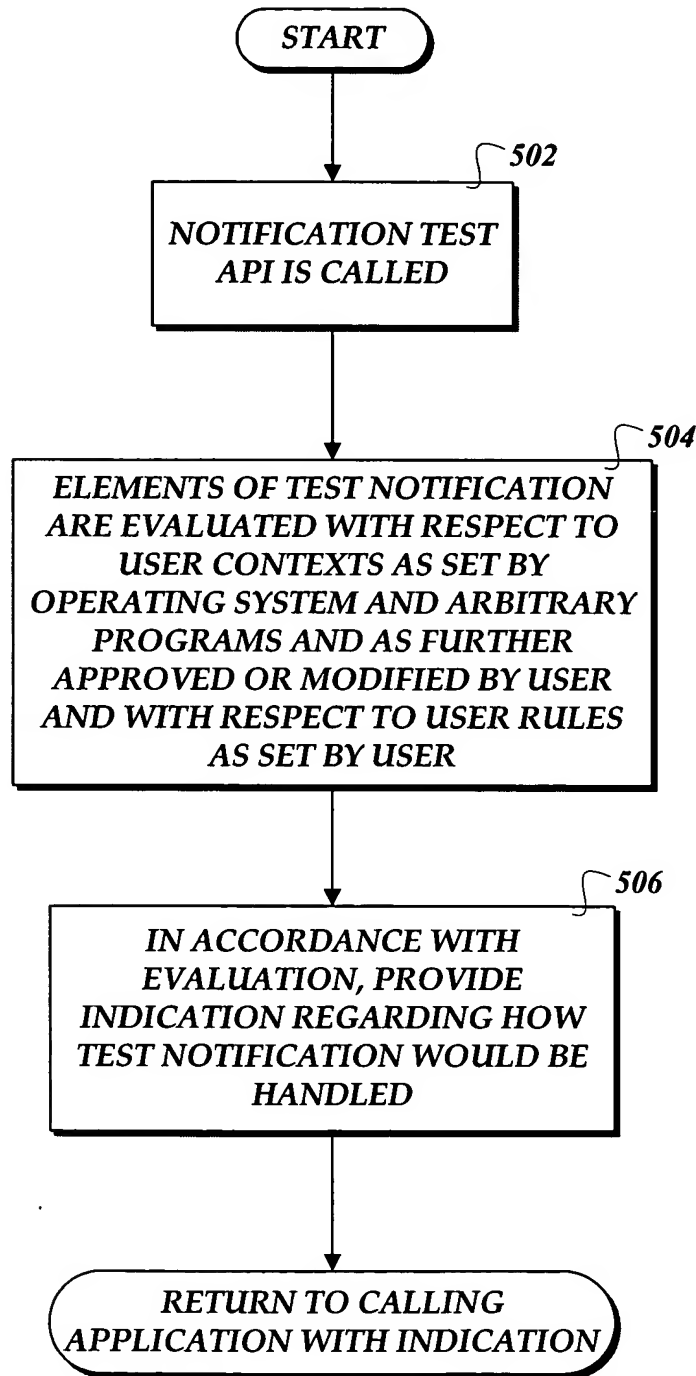
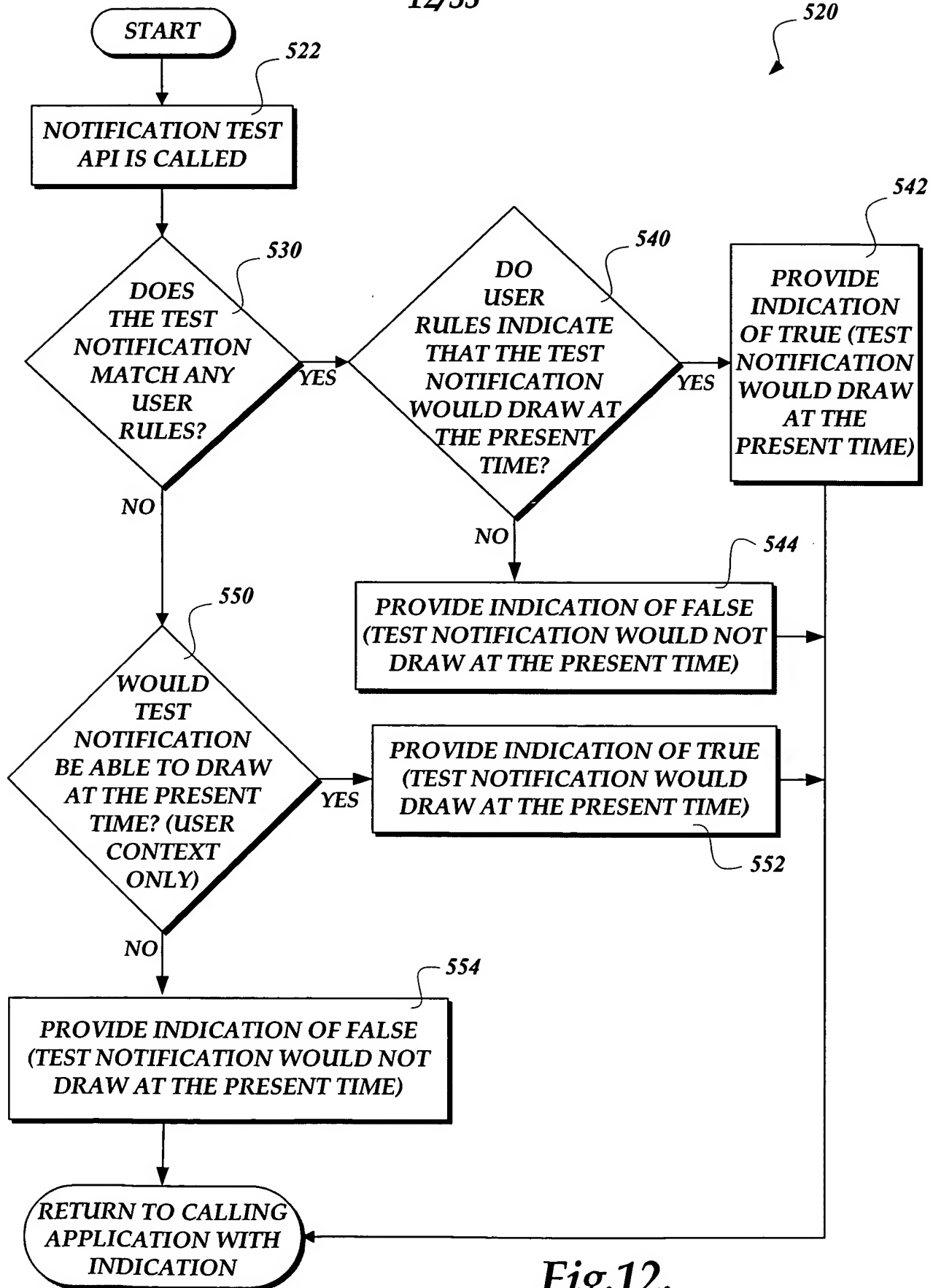
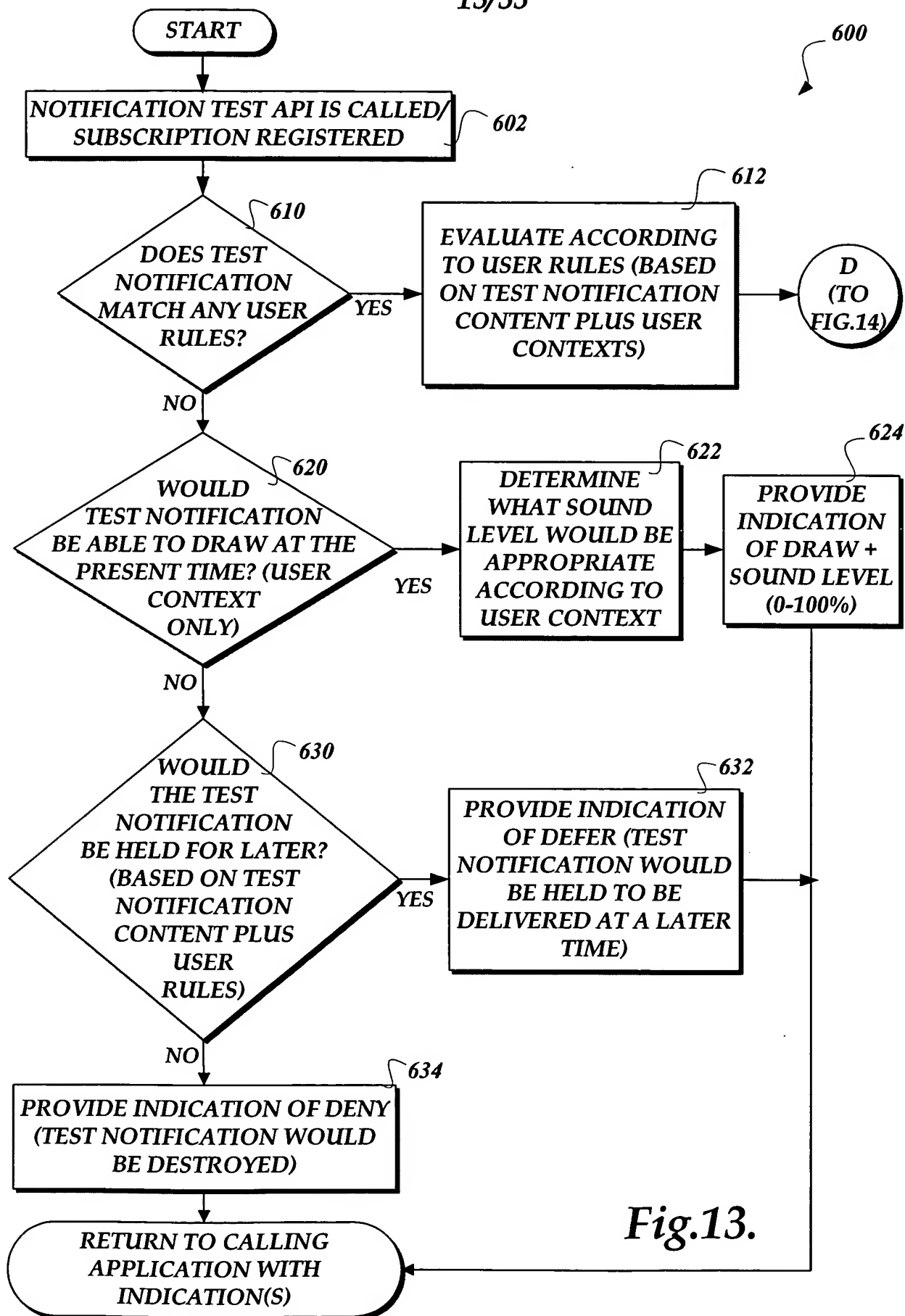


Fig.11.

12/33



13/33



14/33

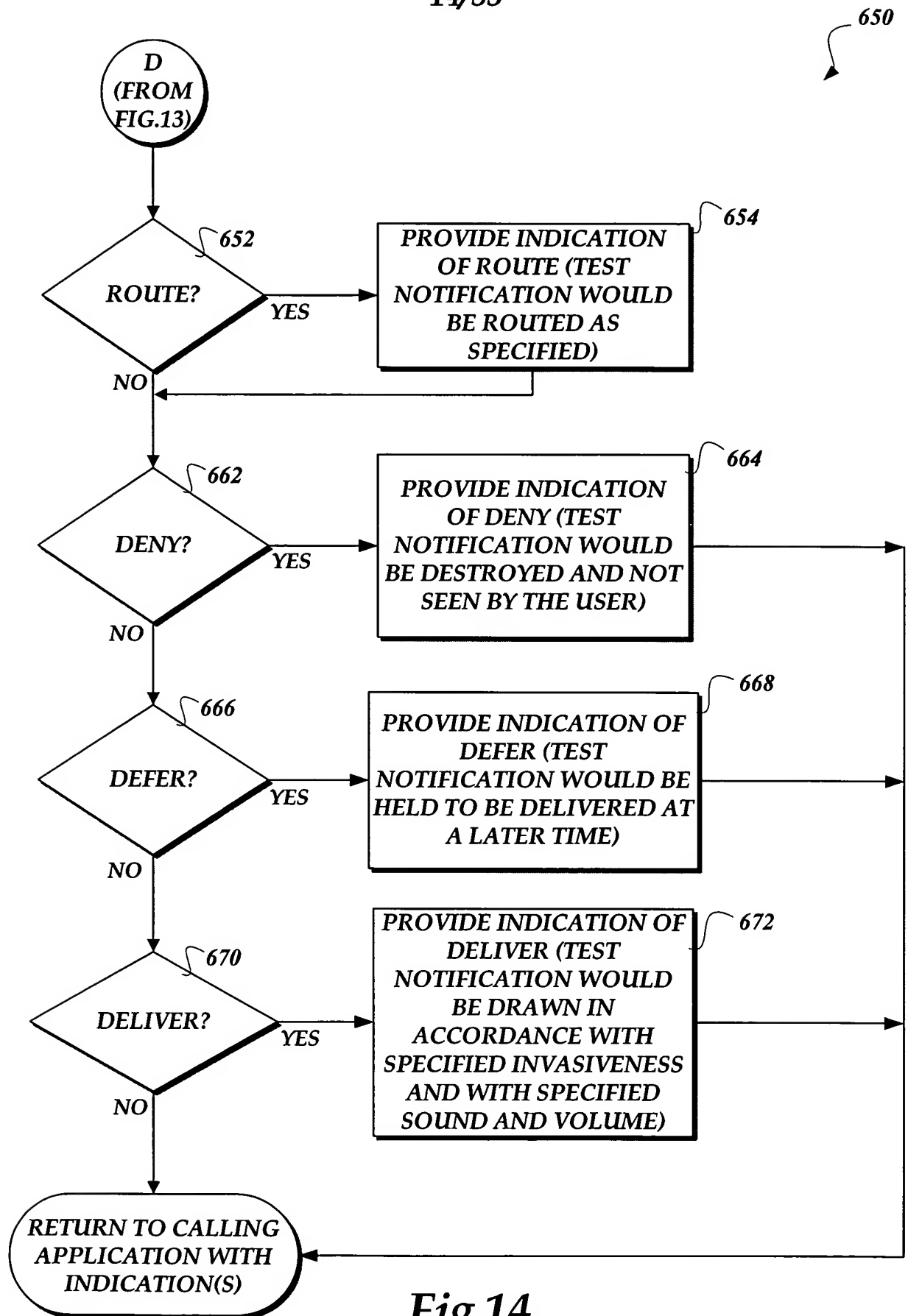


Fig.14.

15/33

1500A

/// <ExternalAPI/>

1510

/// <summary>
/// Use this object to send a balloon Notification to the user. You can derive
this class if you need to add custom properties to this object so that you can have more
context when a NotificationClickedEvent arrives.

/// </summary>

public class Notification

{

public:

Notification();

1520

/// <summary>

/// This is the icon for your notification. Minimum size is 16x16, and
maximum size is 80x80.

/// </summary>

property ImageData HeaderIcon; get, set

1530

/// <summary>

/// This is the title text for you notification.

/// </summary>

property String HeaderText; get, set

1540

/// <summary>

/// This is the main body text of your notification.

/// </summary>

property String BodyText; get, set

1550

/// <summary>

/// Set this property to true if you want the user to be able to click inside a
notification to trigger some event.

/// </summary>

property bool IsBodyClickable; get, set

void AddButton(NotificationButton button);

event NotificationClickedEventHandler

NotificationClickedEvent;

Fig. 15A.

16/33

1500B

1560

```
/// <summary>
/// Call this method after setting all the relevant Notification properties
when you are ready to send this so that it will be shown to the user. This method
returns immediately.
/// </summary>
void Send();
```

1570

```
/// <summary>
/// Use this to update a notification that has already been sent. You can
change some properties and then call Update() to have the changes be reflected.
/// </summary>
void Update();
```

1580

```
/// <summary>
/// Call this if you want to know if a particular notification would actually
draw on screen at this time.
/// </summary>
bool WouldNotificationShow();
```

1590

```
/// <summary>
/// Call this when you want to recall a notification that was already sent,
but that has become obsolete.
/// </summary>
void Revoke();
};
```

Fig. 15B.

17/33

1600

```
/// <ExternalAPI/>
/// <summary>
/// This class helps you manage your context changes. Your custom contexts
are defined in your application manifest. This class lets you set your context to true, or
reset it to false. An example of context is the FullScreen context (that context is
managed by the operating system) which is true when any application is full screen.
The default value for any context is always false.
/// </summary>
public class NotificationContext
{
    public:
        /// <summary>
        /// Construct a context object for a particular context, identified by its
Guid.
        /// </summary>
        NotificationContext(Guid ContextGuid);
        /// <summary>
        /// Change the value of the context. All contexts for an application will be
reset to false when the application terminates or dies.
        /// </summary>
        property bool Context; set only
};
```

1610

1620

1630

Fig. 16.

18/33

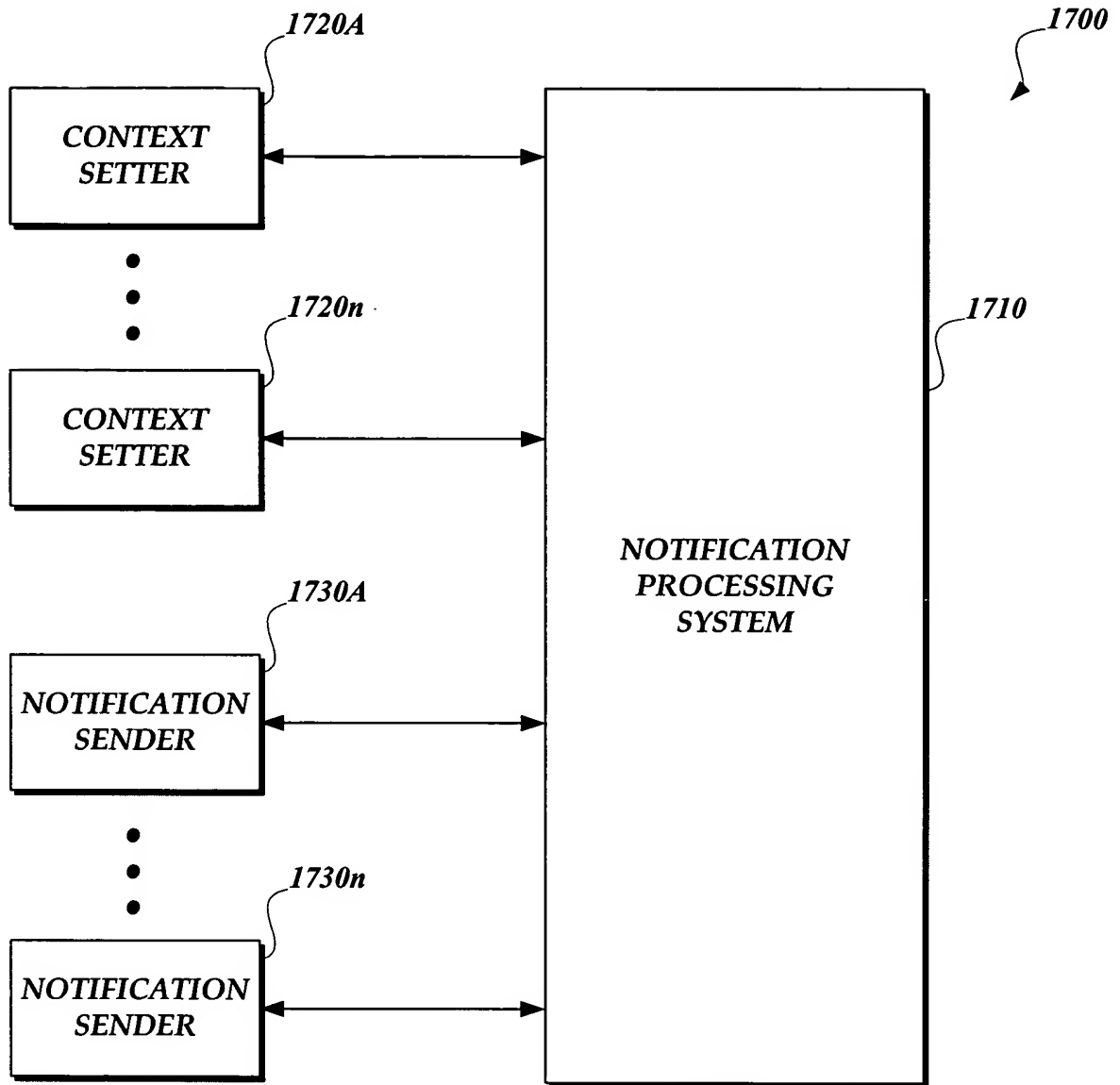


Fig. 17.

19/33

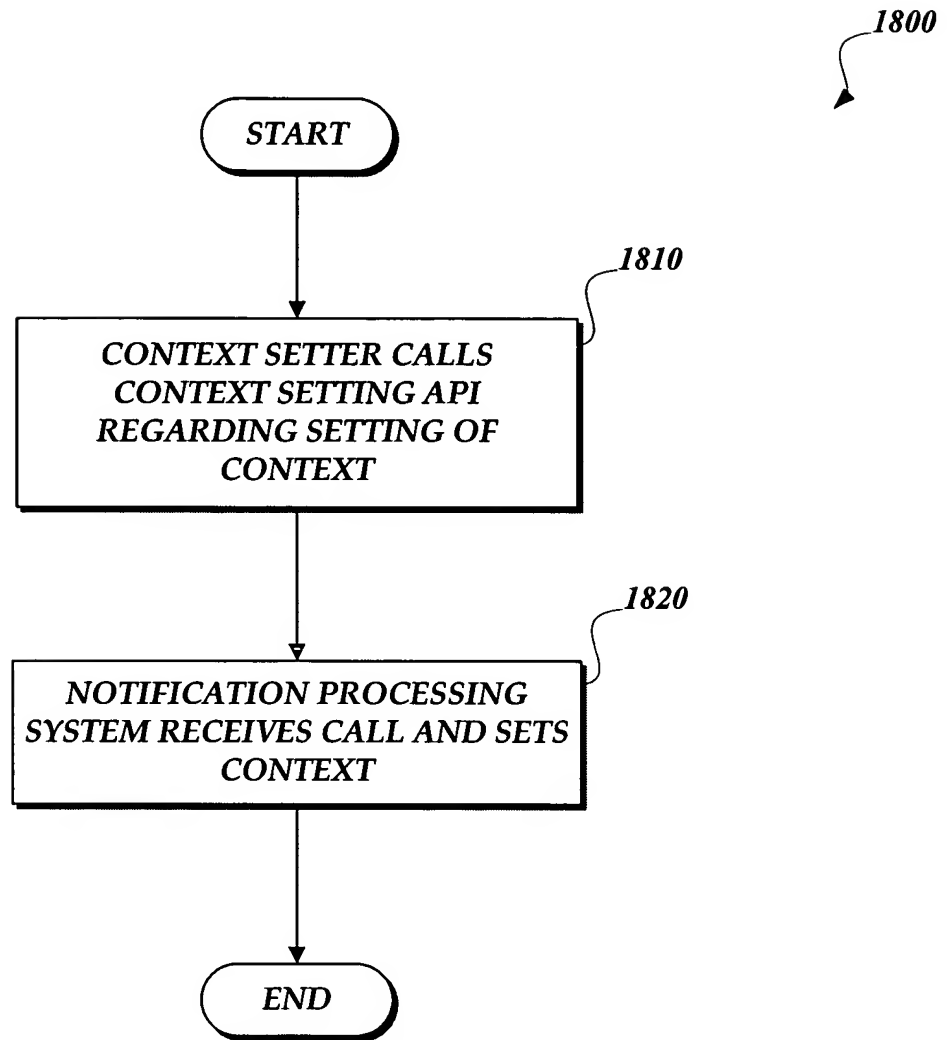


Fig. 18.

20/33

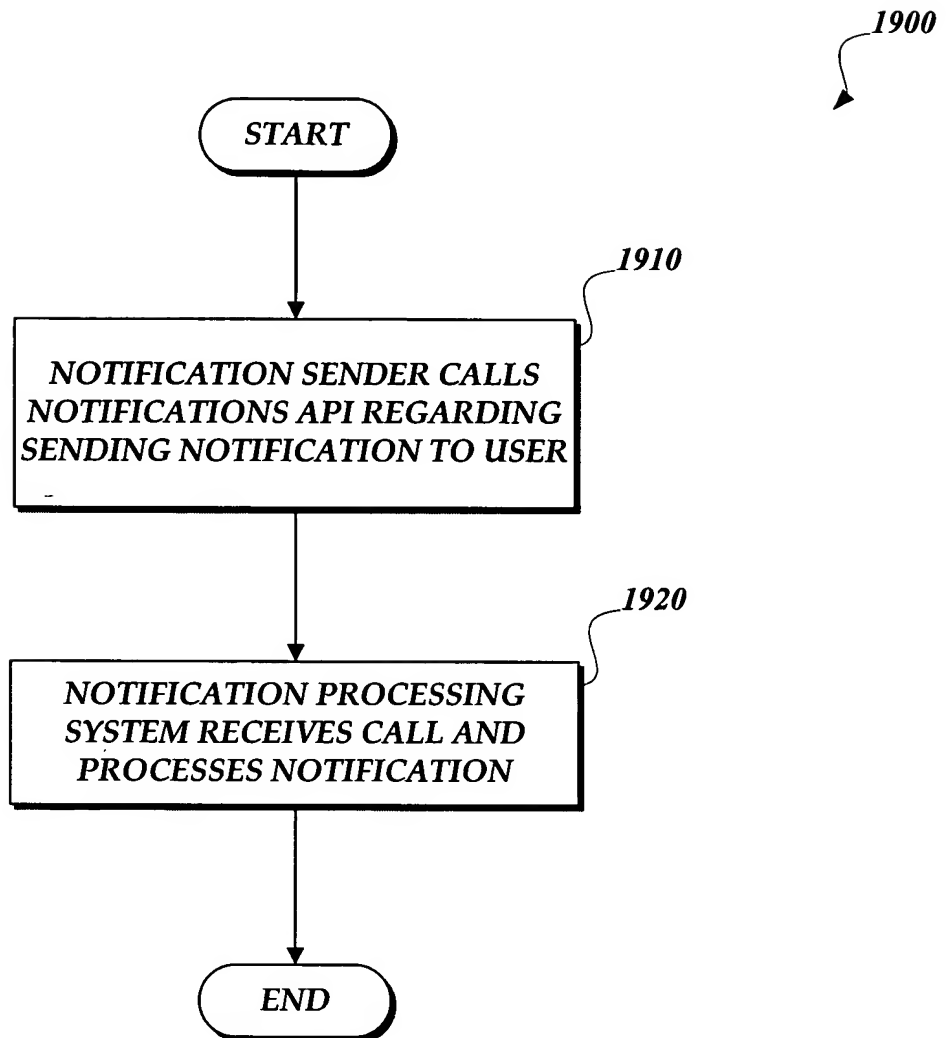


Fig. 19.

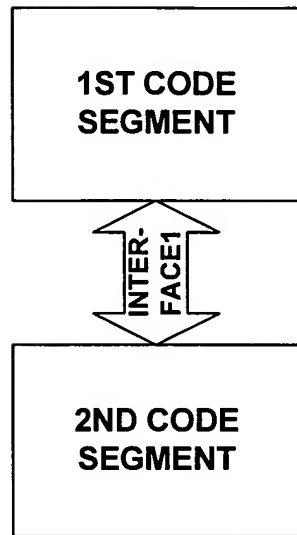


Fig. 20A

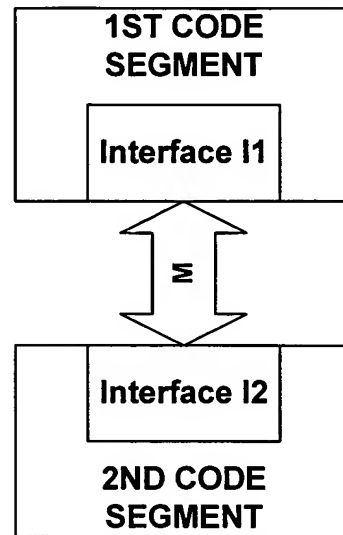


Fig. 20B

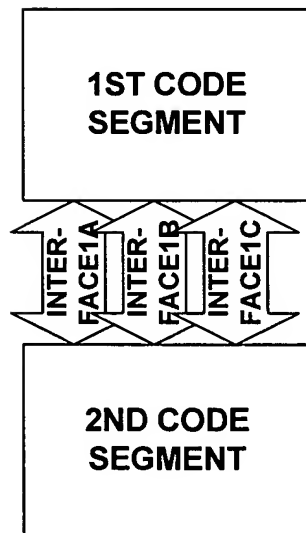


Fig. 20C

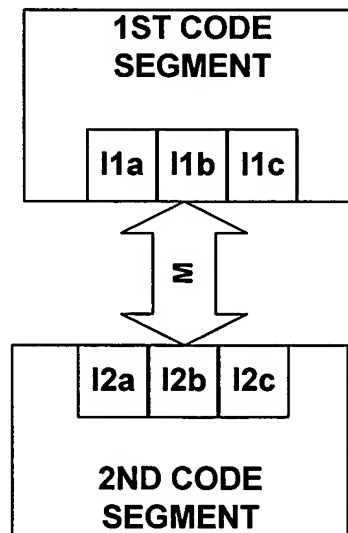


Fig. 20D

22/33

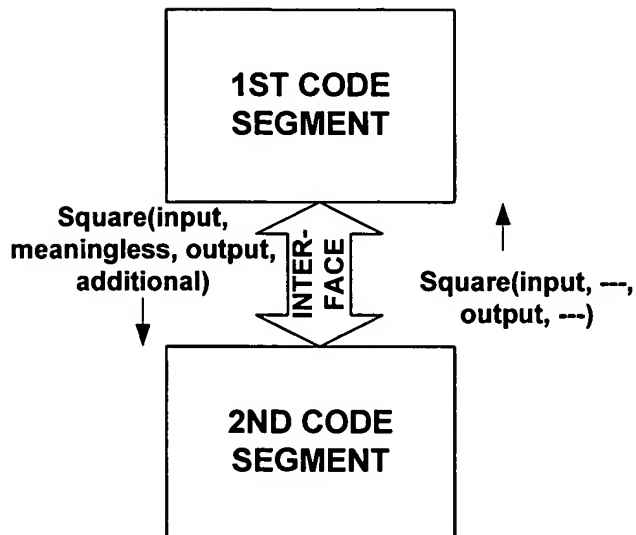


Fig. 20E

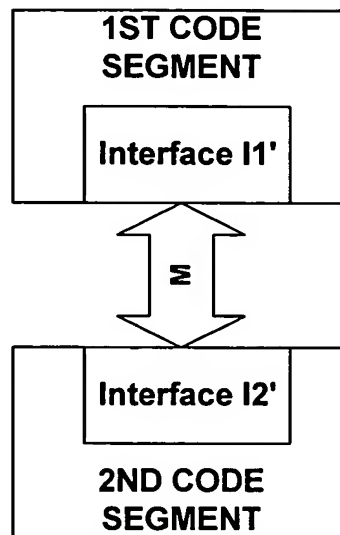


Fig. 20F

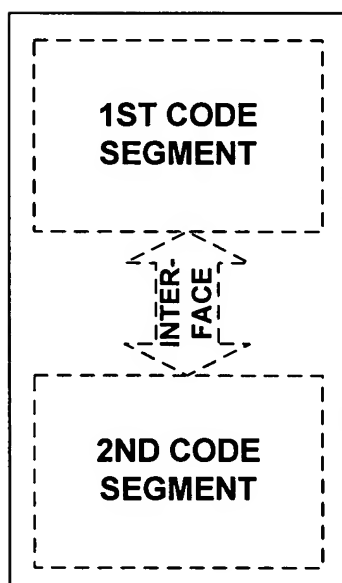


Fig. 20G

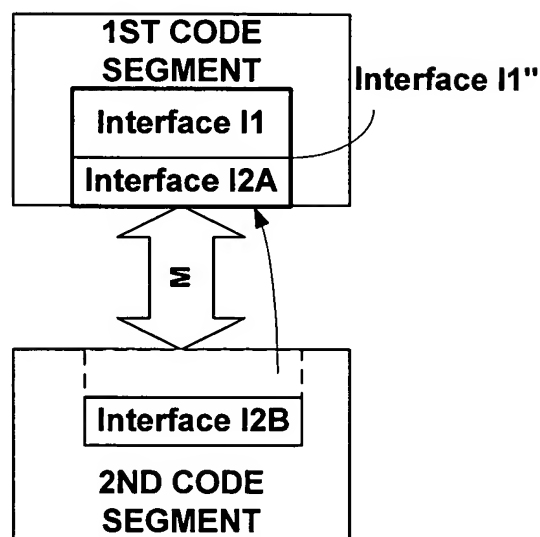


Fig. 20H

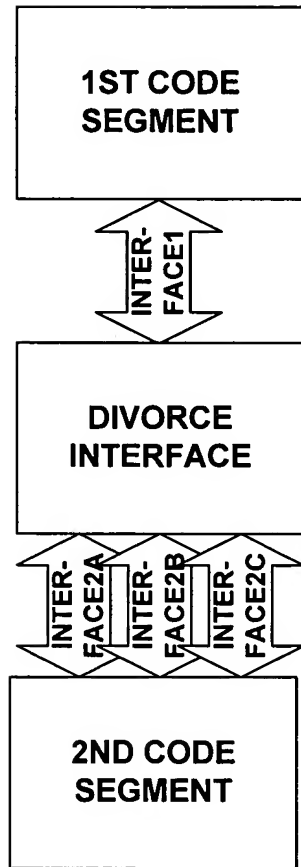


Fig. 20I

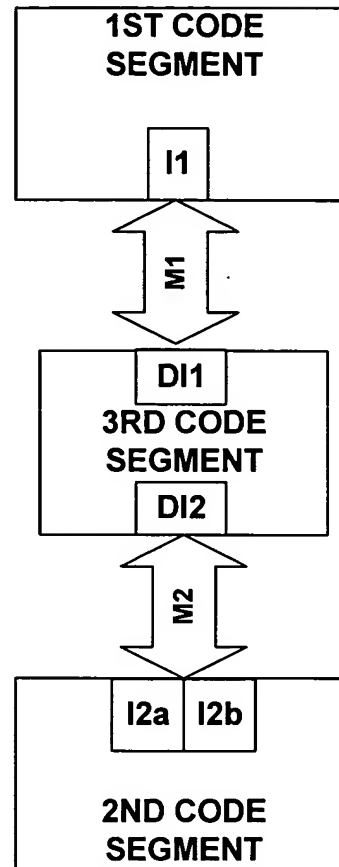


Fig. 20J

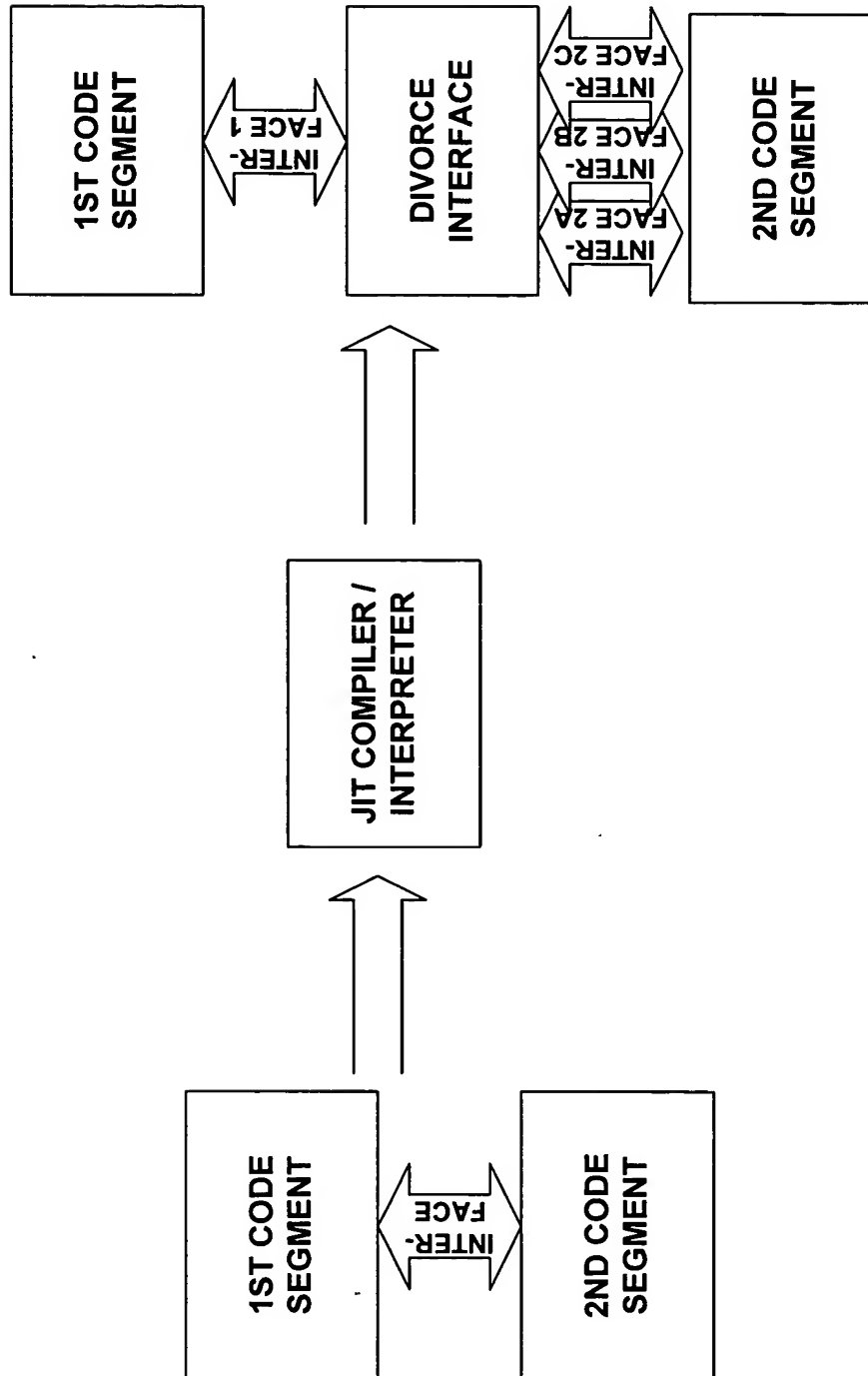


Fig. 20K

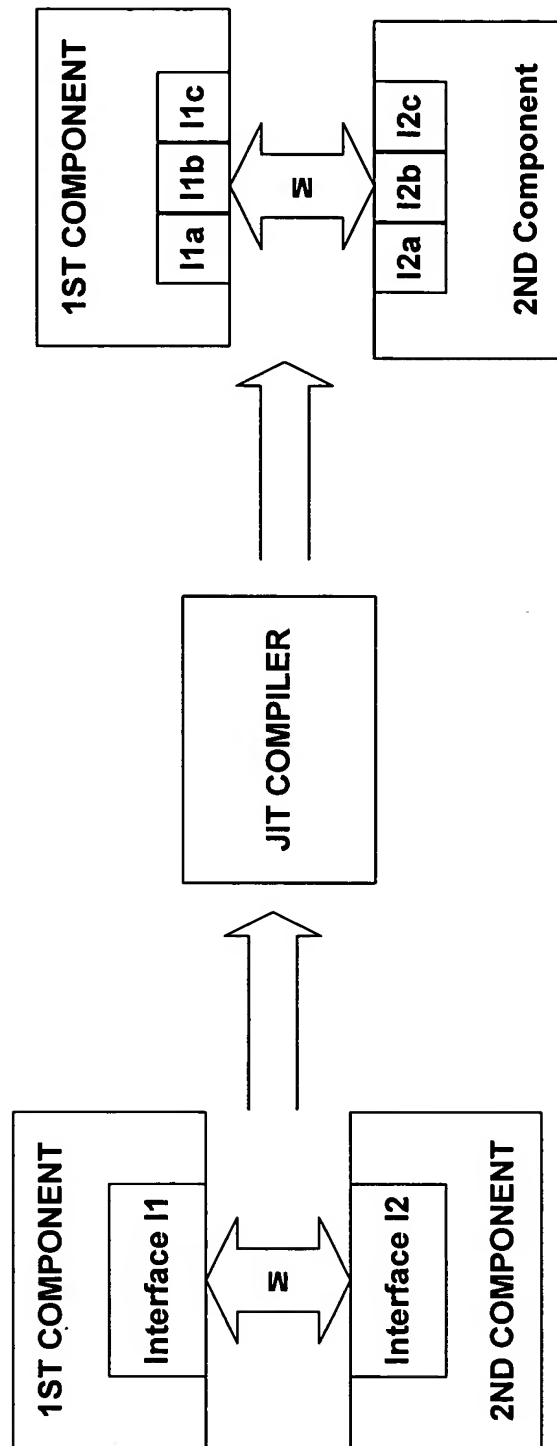


Fig. 20L

26/33

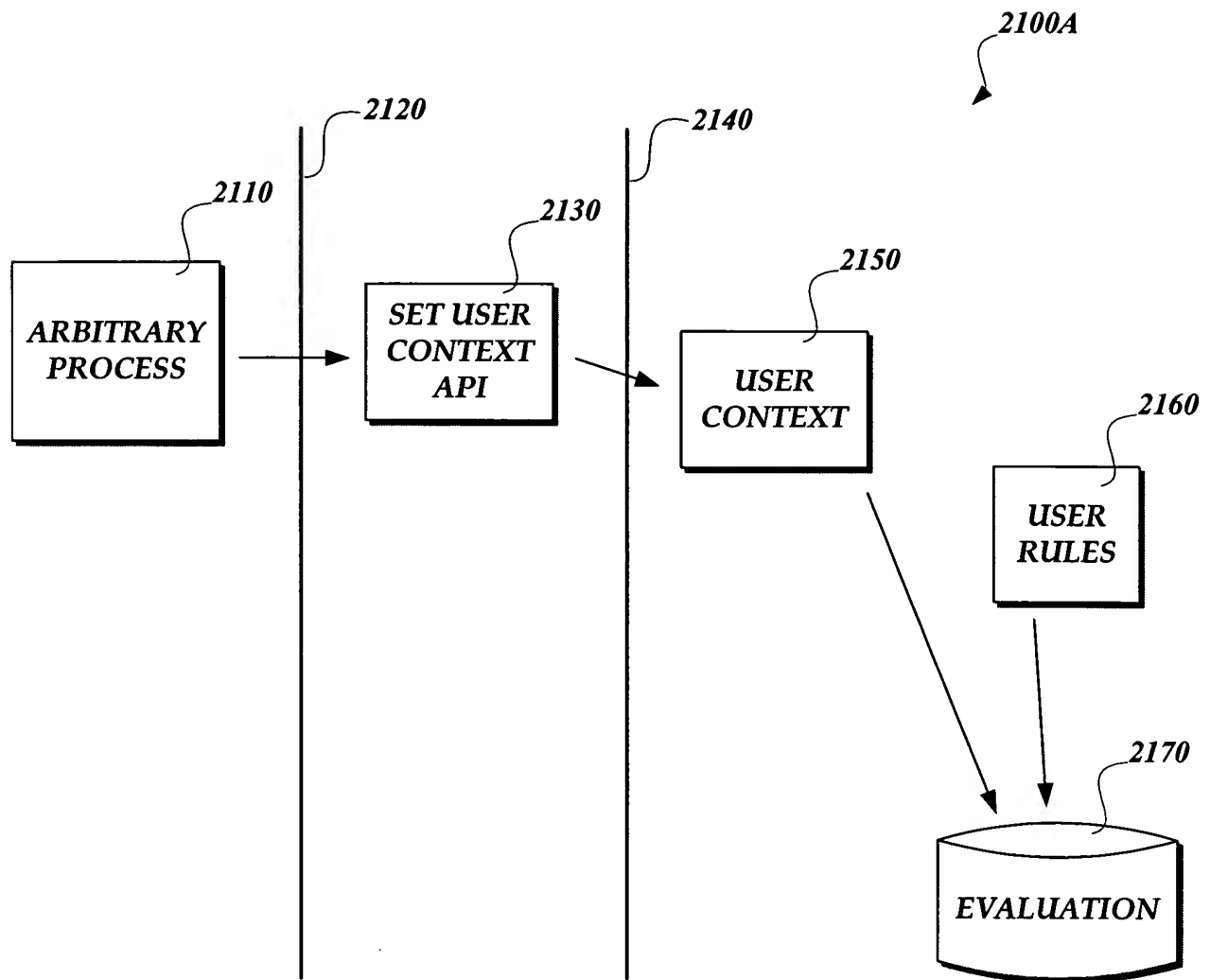


Fig. 21.

27/33

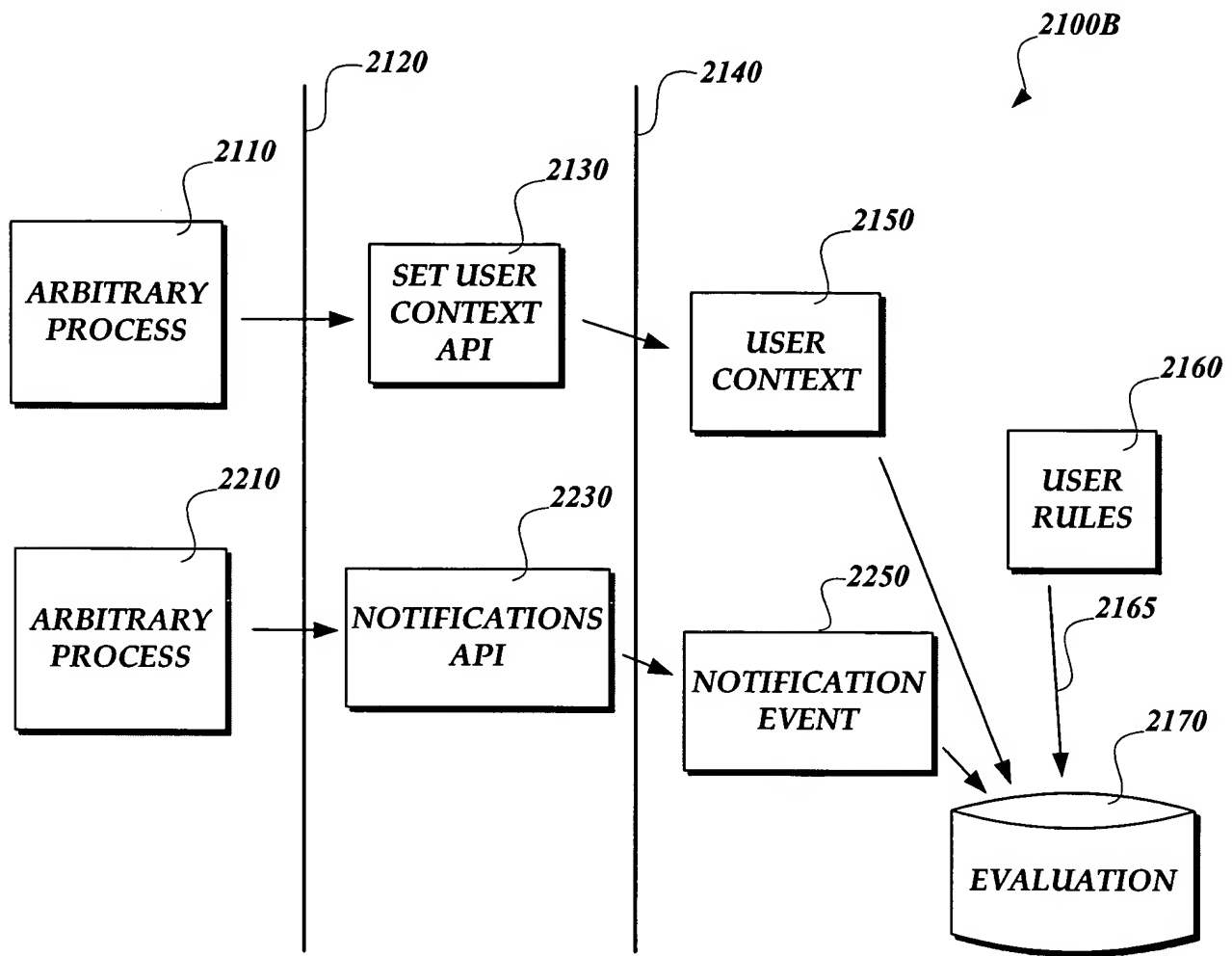


Fig. 22.

28/33

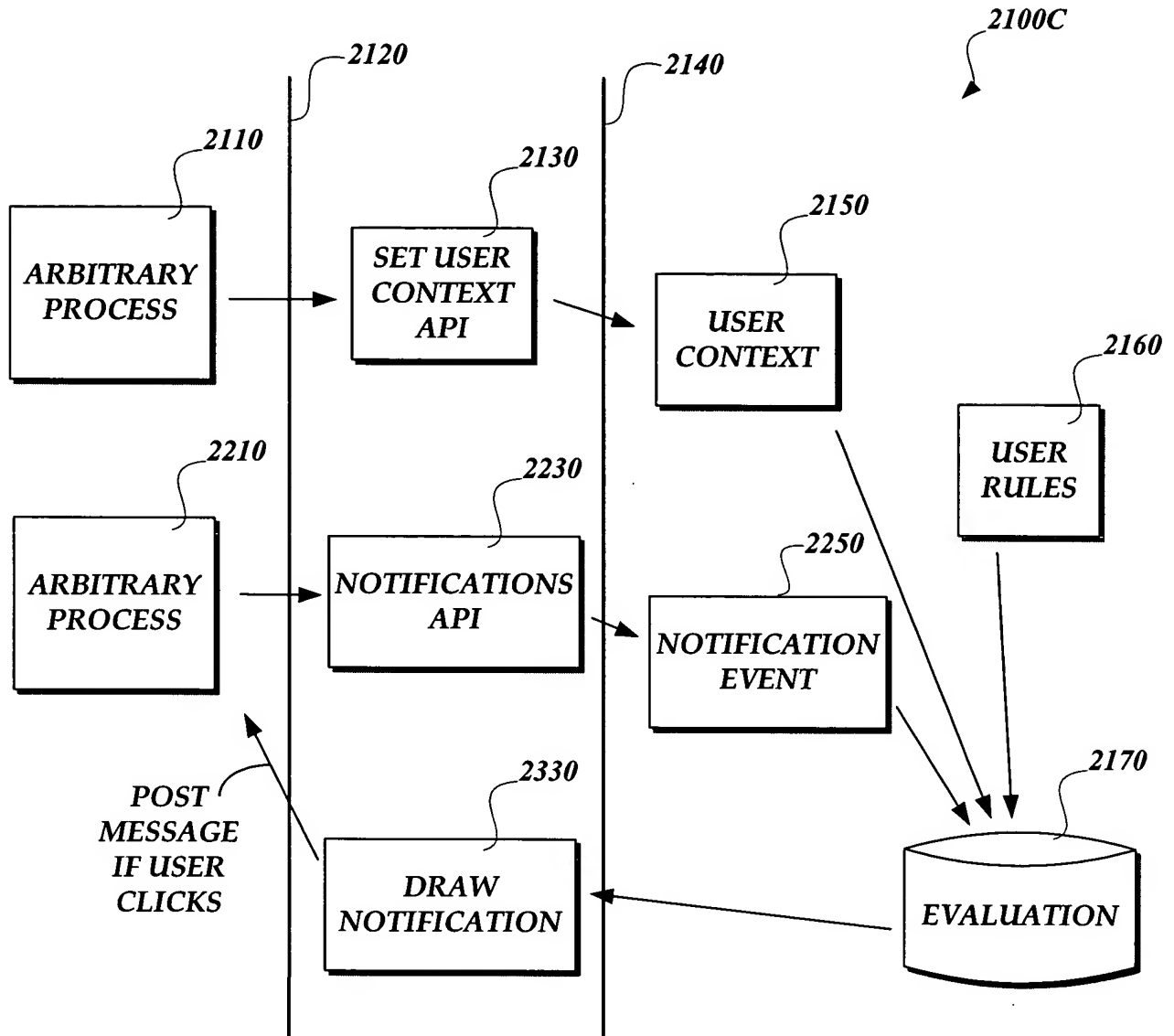


Fig. 23.

29/33

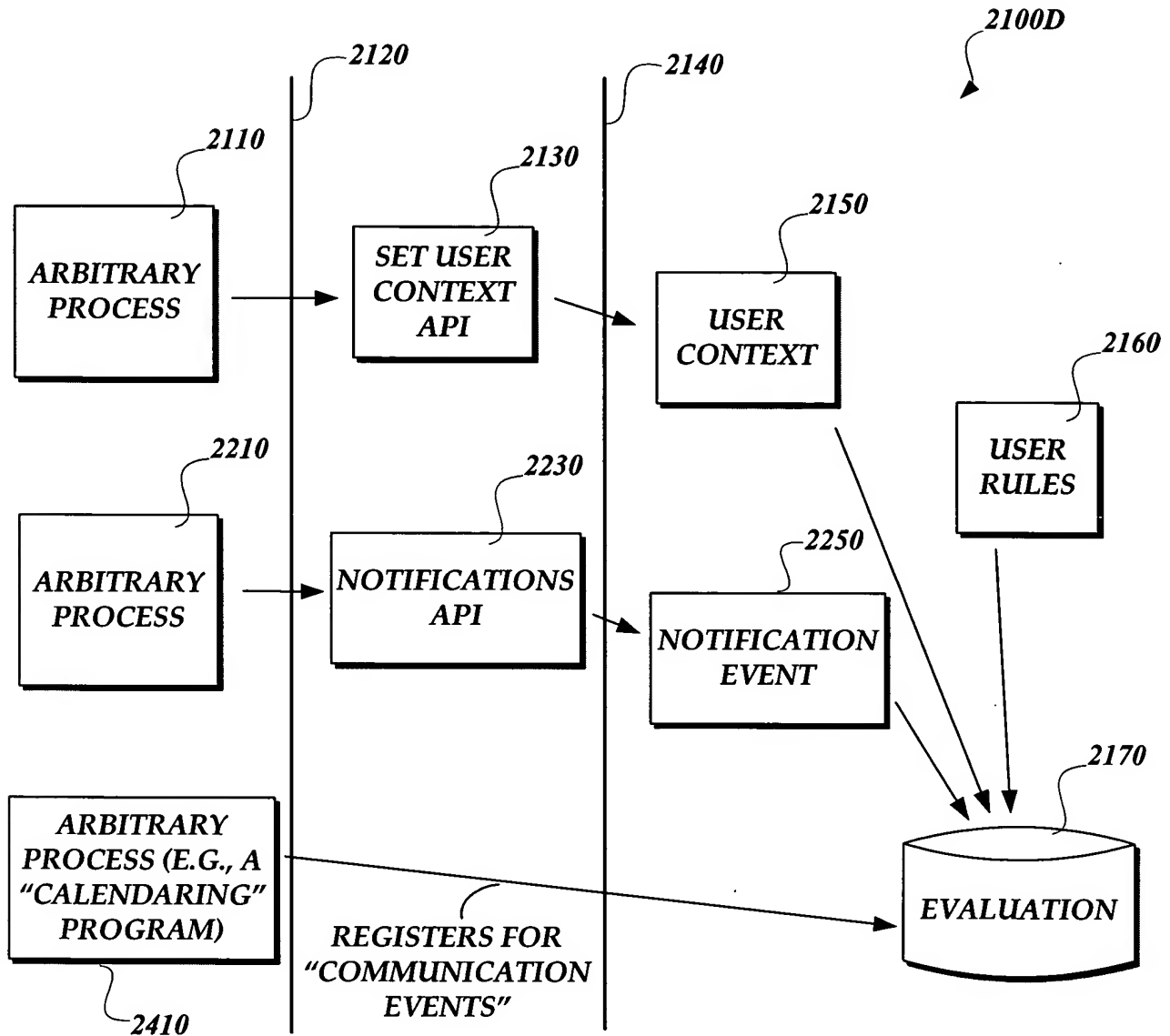


Fig. 24.

30/33

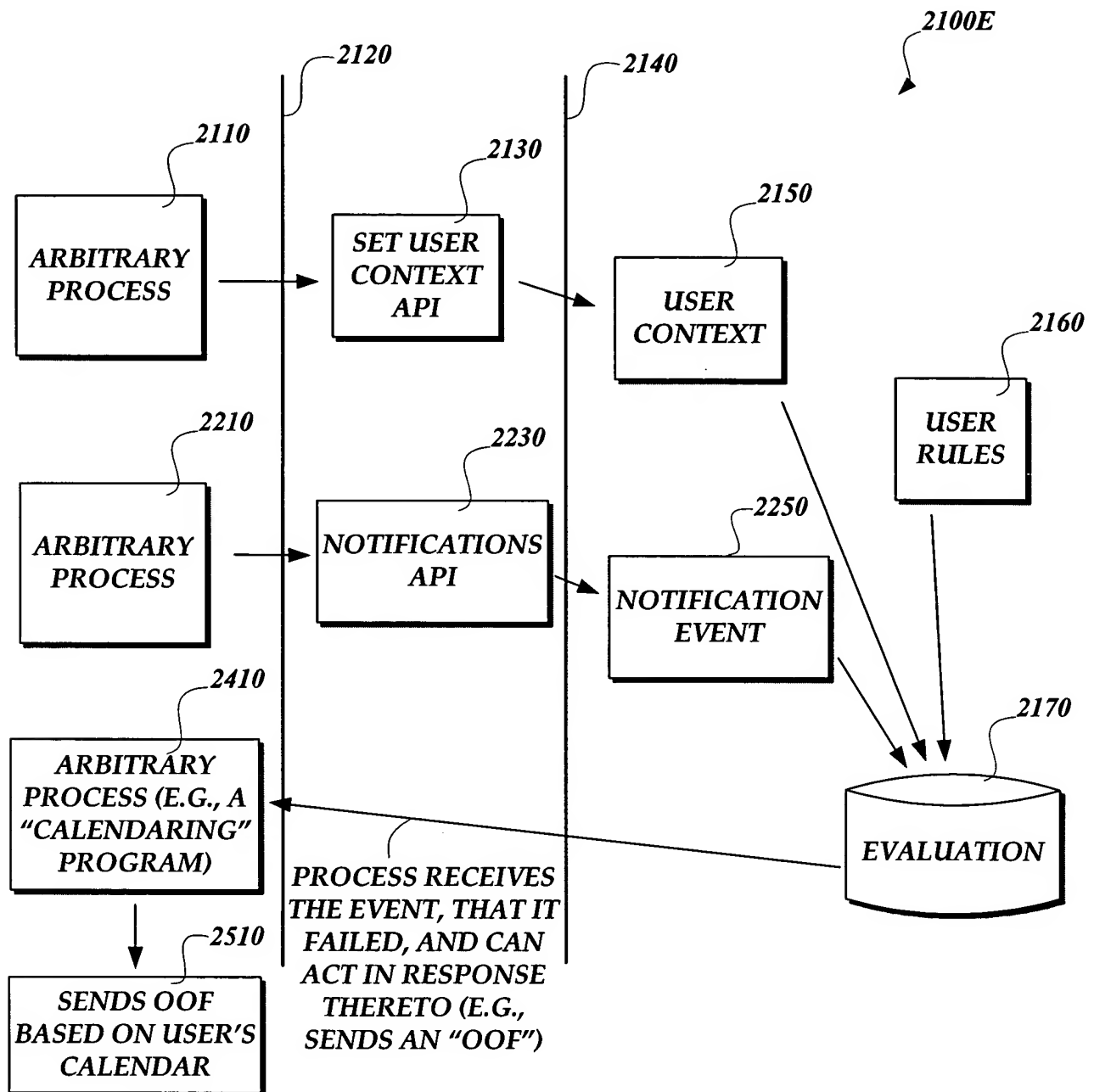


Fig. 25.

31/33

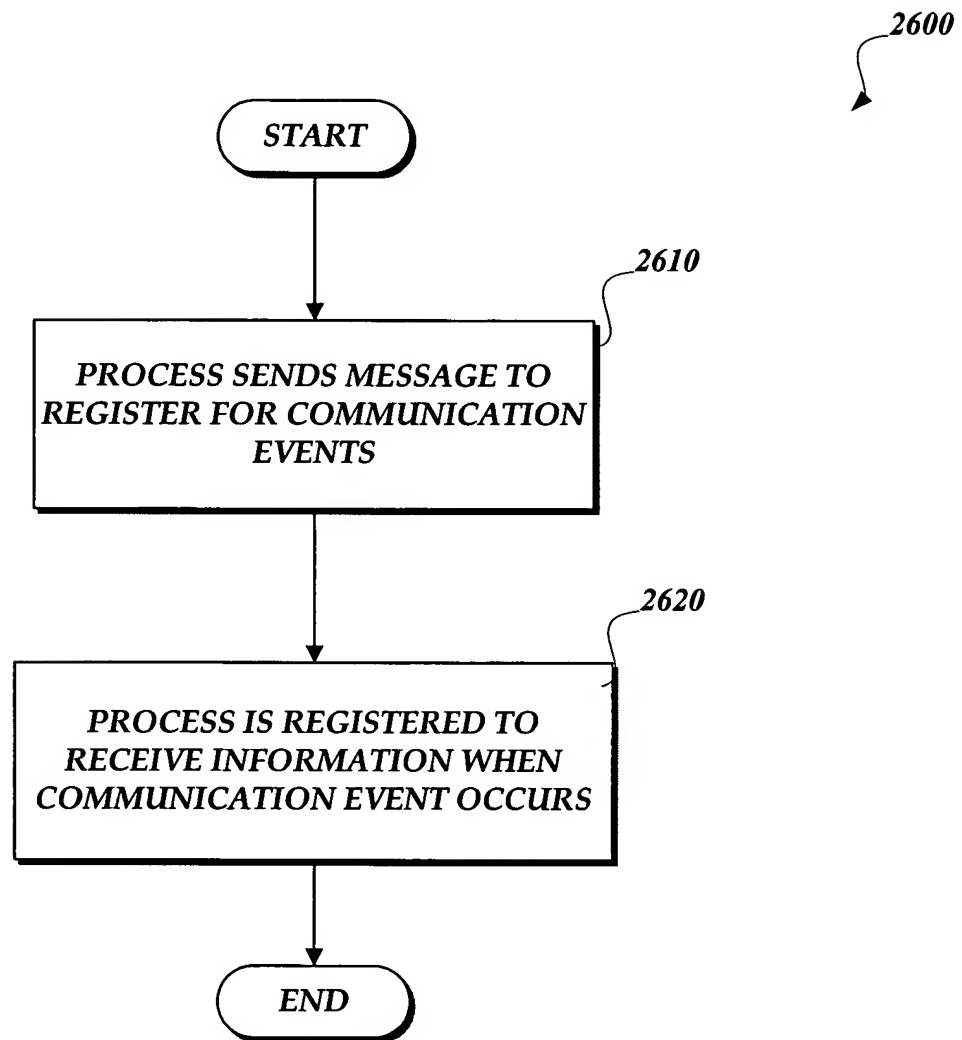


Fig. 26.

32/33

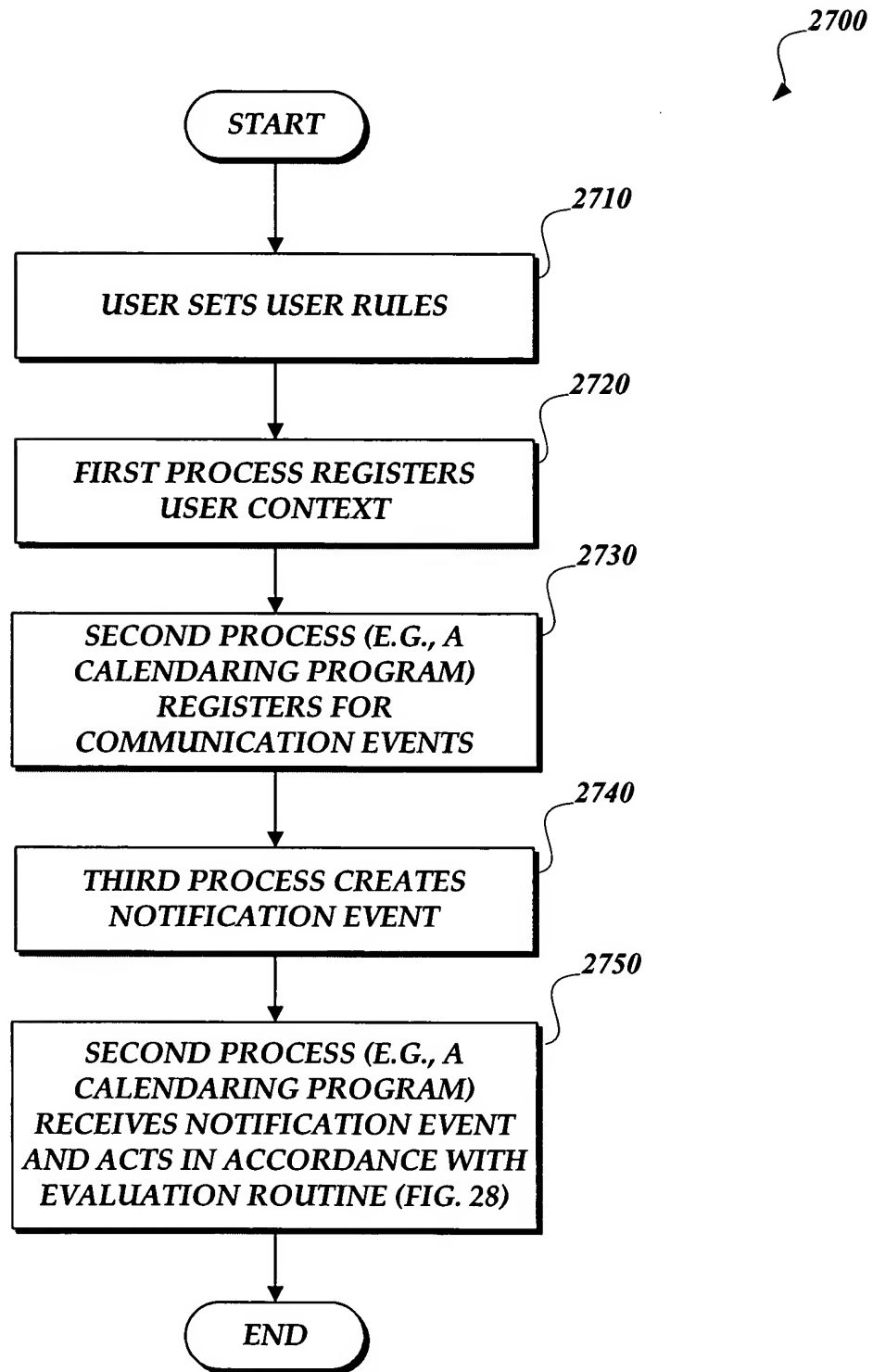


Fig. 27.

33/33

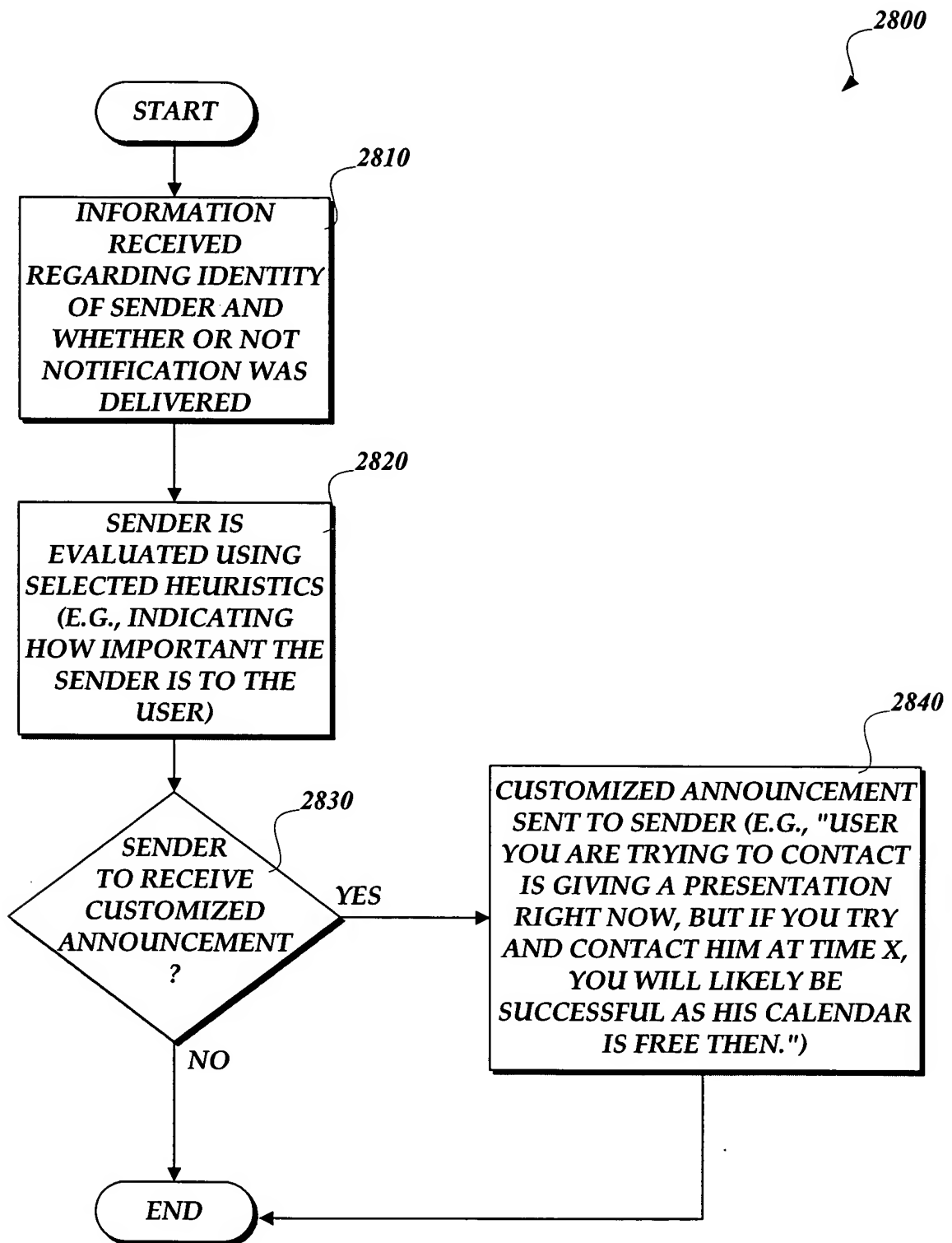


Fig. 28.